CS4100 Computer Architecture

Homework 4-2: Pipelined CPU

Due: 23:59, 6/16 2020

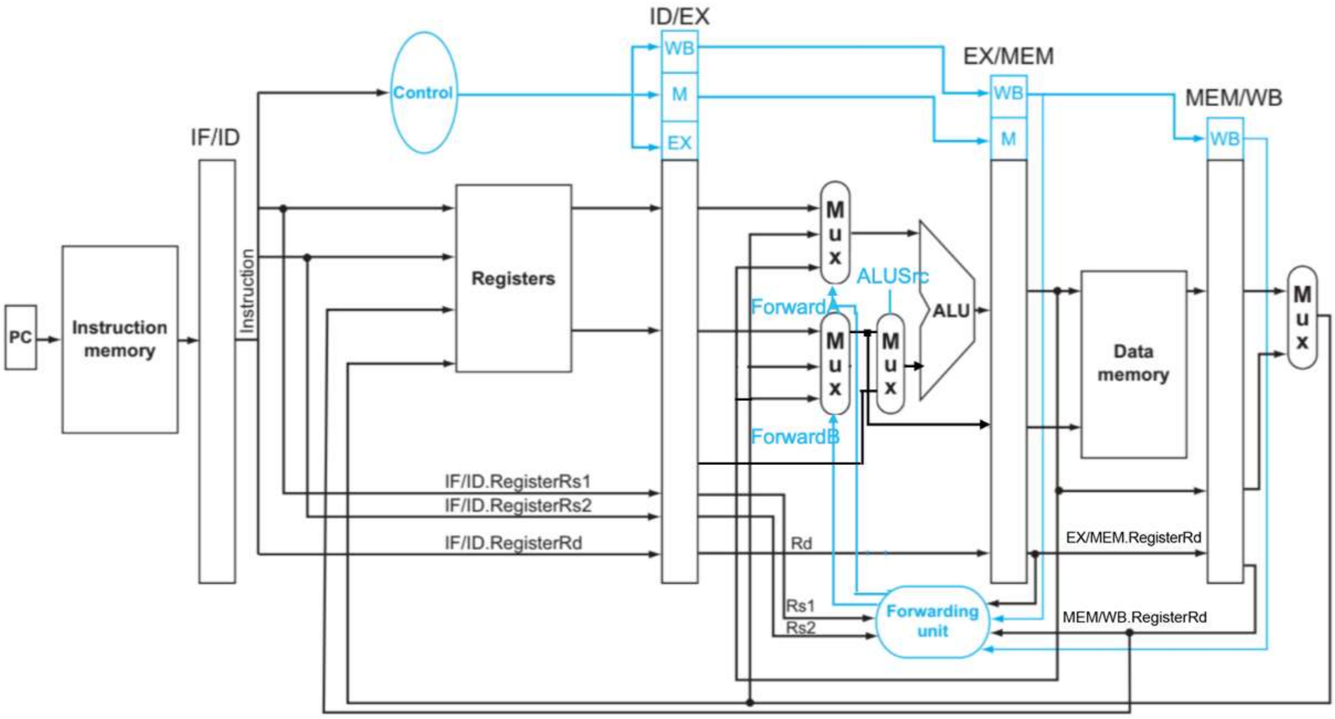
1. Introduction

Compared to a single-cycle CPU, a pipelined CPU is more efficient in using hardware resources. To achieve the efficiency, it divides each instruction into several stages, and leaves spare hardware resources to the following instructions. However, if an instruction uses a register updated by a previous instruction, something wrong might happen. Luckily, by using a forwarding unit to check the data dependency, we can easily avoid that tragedy.

In this homework, you are asked to revise your single-cycle CPU into a pipelined CPU with a forwarding unit as shown below.

2. Problem Description

(1) Architecture



(Note that the sign-extended immediate, ALU control and branch logic are not shown. You have to finish them with the architecture pictures in lecture PPTs.)

(2) Input:

A sequence of instructions in binary is given as the input. The instruction set contains ADD, SUB, AND, OR, SLT, SLTI, ADDI, LD, SD and BEQ. To simplify the implementation, instruction BEQ will be always "not taken", and instruction LD will not cause hazard problems.

Assembly code	Machine code					
ADD	0000000	rs2	rs1	000	rd	0110011
SUB	0100000	rs2	rs1	000	rd	0110011
AND	0000000	rs2	rs1	111	rd	0110011
OR	0000000	rs2	rs1	110	rd	0110011
SLT	0000000	rs2	rs1	010	rd	0110011
SLTI	immediate[11:0]		rs1	010	rd	0010011
ADDI	immediate[11:0]		rs1	000	rd	0010011
LD	immediate[11:0]		rs1	011	rd	0000011
SD	imm[11:5]	rs2	rs1	011	imm[4:0]	0100011
BEQ	imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011

(3) Predefined files (do NOT modify them):

The files below are given to you, but you don't need to hand them in. TAs will use the default version to verify your implementation's correctness. In other words, any modification will NOT take effect.

i. Data_Mem.v
ii. Instr_Mem.v
iii. Pipe_Reg.v
iv. Program_Counter.v
v. Reg_File.v

(4) Predefined files (TODO):

The files below are templates. Their input and output ports are predefined, and what you have to finish are the computations in each module and the connections among the modules.

1.	Adder.v	2.	ALU.v
3.	ALU_Ctrl.v	4.	Control.v
5.	MUX_2to1.v	6.	MUX_3to1.v
7.	Shift_Left_One_64.v	8.	Imm_Gen.v
9.	Forwarding_Unit.v	10.	Pipe_CPU.v (top module)

(5) Output and testbench:

By using `makefile` to run all your Verilog files with `testbench.v`, the console will show all the registers' values in the end. We will check the clock cycle counts of your pipelined CPU to make sure the CPU is a pipelined version.

3. Language/Platform

(1) Language: Verilog

(2) Platform: Unix/Linux

4. Required Items

Do NOT compress your files! Please upload the 10 .v files (TODO) to iLMS.

1.	Adder.v	2.	ALU.v
3.	ALU_Ctrl.v	4.	Control.v
5.	MUX_2to1.v	6.	MUX_3to1.v
7.	Shift_Left_One_64.v	8.	Imm_Gen.v
9.	Forwarding_Unit.v	10.	Pipe_CPU.v (top module)

Do NOT add/modify any clock or rst settings in your Verilog code, for example, #delay.

Do NOT modify any filename.

Do NOT use $stop in your Verilog code.

Any violations will cause strong penalties!

5. Grading

- ✓ 60%: Public testcases. Note that you will get 60 points if you pass all the public testcases; however, if you fail in any of them, you will NOT get any points. In short, please make sure your Verilog code's correctness in the public testcases.
- ✓ 40%: Hidden testcases.
- ※ (Final score) * 0.9: Any interruption during the simulation.
- ※ (Final score) = 0: Plagiarism. We encourage the discussions on the architecture or the data flow rather than code. Don't read others' code before you finish yours!